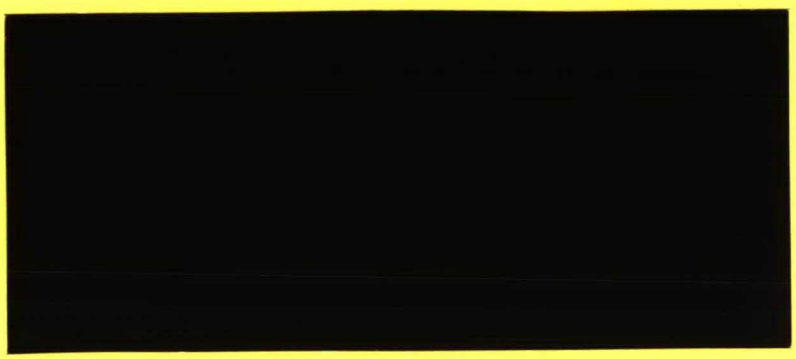
# Applications of constraint logic programming to asset & liability management

Broek, J.M.

*Publication date:*
1989

*Document Version*
Publisher's PDF, also known as Version of record



ITK MEMO

ITK

INSTITUTE FOR LANGUAGE TECHNOLOGY AND ARTIFICIAL INTELLIGENCE INSTITUUT VOOR TAAL- EN KENNISTECHNOLOGIE

I.T.K. Research Memo no. 2
September 8, 1989



# Applications of Constraint Logic Programming to Asset & Liability Management

Johan M. Broek



Institute for Language Technology and Artificial Intelligence,
Tilburg University, The Netherlands

# Contents

# List of Figures

# Introduction and Summary

This document reports on research that was carried out at Amro and ECRC. It describes how asset & liability management models can be constructed using the constraint logic programming language CHIP.

## 0.1 Introduction

Asset & liability management (ALM) is the management of the structure of a bank's balance sheet in such a way that interest-related earnings, e.g. net interest income, are maximized within the overall risk preferences of the bank's management.

ALM can be viewed as a decision process. An important step in this process is the analysis of exposure data using formal models. The analysis is performed to provide some quantitative rationale and support in the management decision making process.

ALM modeling is a time consuming process and requires specialized knowledge. The complexity of ALM problems and the level of uncertainty are high. The valuation techniques used require a considerable amount of numerical computation. The theory is based on assumptions that are often violated in practice. Simulation models are used to perform so-called 'what-if' analysis.

There are many examples of ALM simulation models, ranging from specialized calculators to sophisticated mathematical programming or econometric models. There is, however, little consensus about what an ALM model should do. In fact, there is little consensus about the ALM theory.

CHIP (Constraint Handling in Prolog) is a Prolog-like programming language extended with sophisticated symbolic and numerical constraint solving techniques.

Development time of ALM models can be reduced using CHIP. CHIP programs are short, easy to read and easy to modify. The flexibility of CHIP makes it possible to answer 'what-if' questions. CHIP also provides facilities for data & model management and presentation of results.

This report describes a prototype ALM model implemented in CHIP. The prototype was developed starting from scratch. A new balance sheet item representation was designed and several new computation rules were tested. The model has been run many times using real-life data from the Amro Bank. The prototype enables us to perform 'what-if' analysis and all kinds of optimizations ('what's best?').

## 0.2 Amro & ECRC

This research was carried out partly at the Amro Bank and partly at ECRC. In this section we introduce the two companies.

### 0.2.1 Amro Bank

Amsterdam-Rotterdam Bank N.V. - Amro Bank for short - is one of the leading banking institutions in the Netherlands. Internationally, Amro Bank occupies a prominent position. It has its own branches in the major financial centers so as to provide optimal support to its corporate customers at home and abroad. Moreover, the Bank is very much involved in large international transactions on money and capital markets.

Amro is engaged in all fields of banking and securities business. It offers more than 180000 corporate customers a full range of commercial services and handles merchant banking transactions on a large scale. Retail operations in the Netherlands provide comprehensive services to more than two million individuals via approximately 760 branches. These services cover a wide range of standardized products as well as advice and recommendations tailored to individual needs. [1]

The ALM department of Amro is part of the Group Policy & Services Sector. The main task of this department is to advice the Asset & Liability Committee.

### 0.2.2 ECRC

The European Computer-Industry Research Centre is a joint research organization founded on the initiative of three major European manufacturers: Bull, ICL and Siemens. The Centre is intended to be the breeding ground for those ideas, techniques and products which are essential for the future use of electronic information processing.

The aim of ECRC is to develop fundamental know-how. The Centre's field of activity covers the technologies needed to improve the process of computer-assisted decision making. The program encompasses the set of techniques, and the necessary software tools or machine architectures to support them, needed to resolve the issues of

- Knowledge description and acquisition, where knowledge comprises both basic information and situation-specific decision rules;
- Reasoning mechanisms which build on the aforementioned knowledge;
- Interaction models and tools to enlarge the involvement of both the user and the computer in the decision-making process.

ECRC is not devoted to specific applications development, and its work is more technology driven than application driven. Obviously, however, ECRC uses the analysis of application domains or of applications to try to comprehend the capabilities of the technology it is investigating as well as its limits.

## 0.3 Chapter summaries

- **Chapter 1** In this chapter we introduce ALM and ALM modeling. Asset & Liability management can be viewed as a decision process. An important step in this process is the analysis of exposure data using formal models. Simulation models are used to study the effects of movements in interest rates. These simulation models are used in a 'what-if' fashion. The modeling process is time-consuming and requires specialized knowledge.
- **Chapter 2** CHIP (Constraint Handling in Prolog) is a constraint logic programming language developed at the European Computer-Industry Research Centre in Munich. Constraint logic programming is a paradigm developed to remedy some of the shortcomings of Prolog.

[1] source: Annual Report 1988.

- **Chapter 3** In this chapter we describe the ALM model that was built using CHIP. The chapter is divided into three sections: formulation, implementation and results.
- **Chapter 4** The conclusions of this research, including an evaluation of CHIP as an ALM modeling language, are presented.
- **Appendix A** contains a listing of the ALM model.

# Chapter 1

# Asset & Liability Management

In this chapter we introduce Asset & Liability Management. Asset & Liability management can be viewed as a decision process. An important step in this process is the analysis of exposure data using formal models. Simulation models are used to study the effects of movements in interest rates. These simulation models are used in a 'what-if' fashion. The modeling process is time-consuming and requires specialized knowledge.

## 1.1 Introduction

Asset & liability management (ALM) [1] is the management of the structure of a bank's balance sheet in such a way that interest-related earnings, e.g. net interest income, are maximized within the overall risk preferences of the bank's management [15]. Through their roles as financial intermediaries, banks perform the socially desirable function of improving the efficiency of financial markets. These intermediation activities create profitable opportunities, but they can also introduce interest rate risk (IRR) into balance sheets and income statements of banks [30]. Interest rate risk is defined as the risk that the level of future net interest income (NII) will be affected by movements in interest rates.

Several developments in the financial world have increased the need for ALM:

- Increasing competition and, as a result, decreasing margins force banks to become much more selective in allocating resources.
- Financial markets and the consumers of financial services are becoming extremely sophisticated, resulting in newer and more numerous products and opportunities, as well as in an explosion of the number of market participants.
- Decision support has become more productive, and consequently more affordable, as a result of the advances in financial theory, analytical techniques and processing technology [15].

The need to manage interest rate risk has led to the creation of complex and innovative instruments such as options, interest rate swaps, futures and mortgage-backed securities [25]. In addition to the creation of new investment instruments, a whole range of risk control methodologies emerged.

[1] ALM comprises interest rate risk management as well as credit risk management. This report studies the use of a (simulation) model to support interest rate risk management decisions. It does not consider credit risk or credit risk management.

## 1.2 Decision process

ALM can be viewed as a process [10]. This process consists of three major components: (1) gathering of exposure data, (2) analysis and (3) the management decision process.

The task of an ALM department is to gather and analyse exposure data. The exposure data that is needed is dependent upon the type of analysis. The analysis results in an advice to the Asset & Liability Committee (ALCO). The ALCO formulates policy rules concerning the bank's asset & liability structure. The ALCO's decisions are based on information produced by the ALM department and information from other sources such as government regulations, marketing plans and strategic plans.

## 1.3 Strategy

We have defined ALM as the management of a bank's balance sheet in such a way that interest-related earnings are maximized within the overall risk preferences of the bank's management. This is only one possible interpretation of ALM. Not everyone will agree with this definition. In this section we define three possible ALM strategies or definitions. First, a few useful ALM concepts are defined.

**Mismatch** If every asset is funded with a liability of the same term we say that the structure of the balance sheet is matched. If there are, for instance, more short term liabilities than short term assets we call the structure of the balance sheet mismatched. A mismatched AL combination is a combination of assets and liabilities with different terms.

**Spread** The spread of an AL combination is the difference between the interest rate of the asset and the interest rate of the liability.

**Yield curve** The term structure represents the relationship between interest rates and the time to maturity [20]. Although the yield curve is a specific kind of term structure we will use term structure and yield curve as synonyms.

### 1.3.1 Risk minimization

The first strategy tries to minimize interest rate risk. Because banks perform an important function in the financial system and because a sound financial system is a pre-condition for a healthy economy, banks must avoid creating interest rate risk. To avoid risk the bank will try to minimize the mismatch. This means that each asset is funded with a liability of the same term. Changes in the yield curve will have no effect, or just a small effect, on the NII because the spread on a matched AL combination is constant. However, spreads on matched AL combinations are also very small. Moreover, due to increasing competition, rationalization, and automation, spreads on matched AL combinations are decreasing. In other words, this strategy not only rules out interest rate risk, it also minimizes net interest income.

### 1.3.2 Profit maximization

The second strategy is risk seeking. If we assume that the liquidity preference theory holds, we know that, on the average, long term assets (liabilities) have a higher interest rate than short term assets (liabilities). To maximize NII, the bank will increase the mismatch because the spread on a mismatched AL combination is usually high. However, the NII will be very sensitive to changes in the yield curve. The role of banks is to act as financial intermediaries not as bookmakers. Moreover, this method relies heavily on interest rate anticipation.

### 1.3.3 Compromise

The third strategy tries to find a good tradeoff between risk and income. The bank tries to maximize net interest income within the overall risk preferences. This strategy is much more complex than the first two. Evaluating a certain balance sheet position in terms of future net interest income and interest rate risk is difficult because little is known [2] about future interest rates and market developments.

## 1.4 Risk

We have defined IRR as the risk that the level of future net interest income will be affected by movements in interest rates. But what exactly is risk? A decision is said to be subject to risk when there is a range of possible outcomes which could flow from it and when objectively known probabilities can be attached to these outcomes. Risk is therefore distinguished from uncertainty, where there is a plurality of outcomes to which objective probabilities cannot be assigned. This very strict definition of risk does not correspond with our definition of IRR. Because objective probabilities of interest rate scenarios usually do not exist it is perhaps better to call IRR interest rate uncertainty. Forecasting interest rates is not easy. The best result we can hope for is getting the direction of change right!

There is a second definition of risk which is more liberal. Risk is the possibility of meeting danger or suffering harm or loss. Let's translate this to our definition of IRR. Movements in interest rates can affect the level of future NII. In some cases, the effect of the movements in interest rates is so strong that we speak of a dangerous situation. Which means that the 'well-being' of the bank can be affected. The goal of ALM can be formulated as: To maximize NII while avoiding dangerous situations such as liquidity traps or sudden significant movements in NII.

So how do we continue? One of the previous sections about strategy showed that there is a strategy to minimize risk that does not require knowledge about future interest rates. However, the fact that we cannot make an exact forecast of the weather does not mean that we should stay inside. There are a lot of situations we know that will not occur. For instance, a spread of more than three percent on a matched AL-combination it not very likely to occur. Interest rates are not random. If the rates have been low for a very long time it is not unreasonable to expect an increase. If interest rates have started to increase, it is not unlikely that they will increase for some time to come.

We are not able to make very precise interest rates forecasts, we are, however, able to simulate the effect of an increase in interest rates. By so doing we gain insight into the factors that cause IRR. By carefully analyzing interest rates, market developments and balance strategies, it is possible to control interest rate risk.

## 1.5 Models

The ultimate purpose of an asset & liability management model is to offer a more complete understanding of the bank's earnings and risk dynamics than can be provided by less formal mechanisms. The ALM model should,

- provide a realistic representation of current exposures to interest rate risk and uncertainty,
- translate these exposures to several risk control accounts, and

[2] Forecasting is hard, especially of the future!

- reveal all feasible asset and liability choices that alter the current exposure of the bank toward the desired exposure.

Though an ALM model is not an accounting system, it should be able nonetheless to realistically reflect the actual timing, magnitude and accounting treatment of all expense and income flows. This type of dynamic modeling requires assumptions about managerial behavior, probable loan and deposit demands and the path taken by interest rates. In a robust modeling environment sophisticated relationships may be expressed to establish the relationships between an asset's or liability's growth and other factors. Such relationships may incorporate external economic data or internal policy.

Roughly, there are three generic types of asset & liability models. The first is the Maturity Gap Approach which is currently used by most banks. The second is the Simulation Approach. The third and newest is the Duration Gap Approach. We do not describe the maturity gap approach and the duration gap approach here. We only discuss some of the (dis)advantages of these methods. Toevs and Haney give a full comparison of these models in [30].

The maturity gap approach is easy to use and the results are easily interpreted. This method makes comparison with other banks possible. However, it also has some serious drawbacks. It does not consider interest rates, so it only describes a small part of the IRR. The maturity gap approach only describes IRR at one point in time.

The duration gap approach does consider interest rates. Instead of using the book values of the several accounts it uses the market value. Sometimes it is difficult to determine the 'market value' of some of the items listed on the balance sheet because they cannot be sold. The duration gap approach is still very new and its value for banks in Europe is still unknown.

Simulation models employ a fundamentally different analytical technique than that used in maturity gap and duration gap models. Simulations produce their results in a forward-looking context, whereas the alternatives produce their results statically.

There are two generic types of ALM simulation models:

- Non-optimization models. These models that require an interest rate scenario, a starting balance, and an AL strategy as input, and output an estimation of net interest income and some other information, such as liquidity and solvability ratios.

- Models that perform some kind of optimization.

Models of the first type are nothing more than specialized calculators. Because these models are very simple they can also be very big and produce relatively precise answers. By systematically changing parameters, interest rate scenarios and AL strategies, the sensitivity of the model results can be investigated. This exercise is called sensitivity analysis or 'what-if' analysis. 'What-if' analysis is not only a useful exercise for determining the quality of the model; it also helps the model-builder or model-user to understand more about the problem.

The second type of model is the optimization model. [3] ALM optimization models can be subdivided into two classes:

- Models based on Markowitz's theory of portfolio selection. This theory assumes that managers utilize risk-averse utility functions. The value of an asset then depends not only on the expectation and average of its return but also on the covariance of its return with the return of all other existing and potential investments. These models concentrate on risk minimization.

- Models which maximize the future stream of profits subject to portfolio mix constraints.

[3] The words optimization and optimal solution are mathematical concepts which refer to the solution of the mathematical model.

Examples of the second type of optimisation model are those shown in [19] and the model that will be presented later on in this report. Optimisation models use mathematical techniques such as integer programming and linear programming to find the 'optimal decision'. See [26] for more details about optimisation methods.

One of the disadvantages of optimisation models is that they usually can only handle one type of constraint. Linear programming models, for instance, assume that all the constraints as well as the objective function are linear. If the problem can be translated into linear constraints without losing important information, linear programming is a good tool. If, however, such a translation is unnatural, i.e. we lose information, then the value of linear programming is limited.

"Of course the decision that is optimal in the simplified model will seldom be optimal in the real world [28]". However, it is not impossible that the optimal solution of the model is a 'good' solution in the real world.

Still we must always test the sensitivity of the model results to erroneous assumptions and imprecise data. We must question the value of the results of the model. How much of reality does the model capture? How precise should the results of the model be? What is the influence of our simplifying assumptions? How sensitive are the results of the model to changes in parameters? To answer all these questions the model must be rerun several times with different (combinations of) assumptions and different parameters. Statistical techniques can be used to help us to evaluate the results of these simulations; see e.g. [18].

The two simulation modeling approaches are complementary. The first type of model can be very big and give relatively precise results. Optimization models can help us to generate solutions and strategies. In both cases sensitivity analysis or 'what-if' analysis is a necessity. There are several problems we encounter when performing sensitivity analysis or 'what-if' analysis:

- Collection of data. Collection of data is always a problem. Exposure data are derived from many decentralized accounting systems. The data usually contain errors and must therefore be checked, which can take some time and effort. The data must be presented to the model in the right format.
- Data & results management. Model data & model results must be stored in such a way that they can be retrieved, in the right format, when necessary.
- Formulation of strategies. Formulation of good strategies to test is a process requiring specialized knowledge. Optimization models can help to form ideas about good strategies.
- Model adjustment. The model must be adjusted very often to be able to run new experiments.
- Model management. ALM models are complementary. Several models can be used to study the same problem. By comparing the results of different models, we gain more insight into the problem.

# Chapter 2

# CHIP

CHIP (Constraint Handling in Prolog) is a constraint logic programming language developed at the European Computer-Industry Research Centre in Munich. Constraint logic programming is a paradigm developed to remedy some of the shortcomings of Prolog.

## 2.1 Constraint logic programming

The combination of relational form, non-deterministic computation and unification in Prolog results in a high-level programming language with a strong theoretical foundation; see e.g. [21, 1]. Prolog is an excellent programming language for writing symbolic computation and natural language programs. When applied to, for instance, Operations Research or Financial Modeling, Prolog shows shortcomings, see [7]. These shortcomings can be summerized as follows.

- Lack of expressive power. Prolog carries out computations in the Herbrand universe. [1] Unification is used to solve equations in this universe, i.e. on uninterpreted terms. When modeling a problem, one must use a mapping from the intended domain (e.g. sets, rational terms) on the Herbrand universe. This causes the loss of both naturalness of problem expression and Oefficiency.

  The problems with negation as failure can be seen as an example of the lack of expressive power of Prolog. It is not possible to express negative information with pure Horn clauses. They state what is true but not what is false.

- Poor control facilities. The poor control facilities of Prolog lead to inefficiency and infinite loops. In Prolog the *control part* of the algorithm is (almost) fixed. Sometimes we want to be able to control the *control part*. This occurs, for instance, when we want to make a sound implementation of negation or when the problem is very complex and a more specialized control is necessary.

Several extensions to Prolog have been proposed to remedy these shortcomings, for instance, functional programming in Prolog, equation solving in Prolog, meta-rules, e.g. METALOG [6], delay mechanisms, e.g. [22, 23], intelligent backtracking etc.. One of the most recent extensions is the introduction of constraint solving techniques into logic programming.

The expressive power of Prolog is limited because it carries out computations in the Herbrand universe. Suppose we want to design systems over well understood domains such as sets, graphs, Boolean expressions, integers, rationals, etc. These domains have natural algebraic operations associated with them such as set or graph intersection, disjunction, and

[1] The Herbrand universe is the collection of all ground terms (informal definition).

multiplication. They also have associated privileged predicates such as set equality, graph isomorphism and various forms of inequalities. These predicates are examples of what we call constraints.

Unification can be seen as a special kind of constraint solving: the unification algorithm decides whether two terms can be made equal. The main idea of constraint logic programming is to replace unification by the more general concept of constraint solving.

The introduction of constraint solving into logic programming has several advantages.

- The constraints state properties directly in the domain of discourse as opposed to having these properties coded into Prolog terms or LISP lists [16].
- The constraints have the ability of representing properties implicitly as opposed to having bindings to variables.
- The constraint solving paradigm enables the use of interesting problem solving techniques like local value propagation, data-driven computation and consistency checking. Constraints can be used in an 'active' way to prune the search-tree.
- Constraint solving can be introduced into logic programming without producing overhead and without changing the declarative aspects of logic programming.

Examples of constraint logic programming languages are CLP(R) [17], CHIP [8], and Prolog III [5].

There are limitations to the types of constraints that can be used in constraint logic programming. These limitations of constraint logic programming, however, are well defined. The Constraint Logic Programming (CLP) Scheme [16] defines a class of languages based upon the paradigm of rule-based constraint programming. Each instance of the scheme is a programming language and is obtained by the specification of structure of computation. That is, the domain of discourse and the functions and relations on this domain characterize the language.

A particular instance of the scheme is CLP(R) [17]. The domain of computation is the set of real numbers and the constraints solved by the constraint-solvers are linear equations and inequations.

Other examples of constraint logic programming languages are Prolog III and CHIP. Prolog III uses a simplex-like algorithm to solve linear equations and inequations and provides a saturation method to deal with Boolean terms [5].

CHIP (Constraint Handling in Prolog) is a language developed at ECRC in Munich. It provides three new computation domains: finite domain restricted terms, Boolean terms and linear rational terms. For each of them CHIP uses specialized constraint solving techniques: Consistency Techniques for finite domains, equation solving in Boolean algebra for Booleans and a symbolic simplex-like algorithm for rationals.

## 2.2 CHIP

### 2.2.1 Finite Domains

One of the main extensions of CHIP is to provide domain-variables, i.e. variables ranging over a finite domain. [12, 14]. CHIP provides a large variety of constraints on domain-variables. The constraints can be symbolic, numerical or user-defined [8]. Listing all of them is outside the scope of this report. We therefore confine attention to the use of linear terms.

### Linear terms

Domain-variables over natural numbers can be used to build a class of terms, called linear terms. These terms are used in some important constraints, for example linear equations and inequalities as well as in higher-order predicates for combinatorial optimization. A linear term is defined in the following way.

1. A natural number is a linear term.
2. A domain variable ranging over natural numbers is a linear term.
3. If $c$ is a natural number and $X$ is a domain variable ranging over natural numbers or $X$ is a natural number and $Y$ is a linear term then $c * X + Y$ is a linear term.

### Constraints on linear terms

These are just a few of the possible constraints on linear terms. If $X$ and $Y$ are linear terms then,

$$X > Y, X \geq Y, X < Y, X \leq Y, X = Y, X \neq Y$$

are well-formed constraints of CHIP.

The example shows a possible application of linear terms.

```
money([S,E,N,D,M,O,R,Y],[R0,R1,R2,R3]) :-
   [S,E,N,D,M,O,R,Y] :: 0..9,
   [R0,R1,R2,R3] :: 0..1,
   R3 = M,
   S ## 0,        % S is not equal to 0
   M ## 0,
   alldistinct([S,E,N,D,M,O,R,Y]),
   R2 + S + M #= O + 10 * R3,  % #= means: is equal
   R1 + E + O #= N + 10 * R2,
   R0 + N + R #= E + 10 * R1,
   D + E #= Y + 10 * R0,
   labeling([R3,R2,R1,R0,S,E,N,D,M,O,R,Y]).

labeling([]).
labeling([X|Y]) :-
   indomain(X),
   labeling(Y).
```

It is the well-known SEND + MORE = MONEY problem. CHIP solves this problem in less than one second. Of course, CHIP was not just designed to solve Artificial Intelligence puzzles. Problems such as scheduling, graph colouring, and car sequencing have been solved using the finite domains; see [9, 12, 13, 14].

## 2.2.2 Booleans

Another extension CHIP provides is Boolean unification. Boolean unification solves symbolically equations over Boolean terms. The Boolean unification algorithm is based on variable elimination. Boolean unification can be used for problems such as formal verification of hardware and circuit simulation [29].

### 2.2.3 Rationals

The third extension, and for our application the most interesting is rational arithmetic. The rational arithmetic part of CHIP has been used to model hybrid circuits [11] and financial applications [2, 3].

#### Linear rational terms

A rational number is a fraction of two multiple precision integers. Rational terms are defined as follows.

1. A rational number, i.e. a rational or integer valued constant, is a rational term.
2. Rational variables are rational terms.
3. If $t_1$ and $t_2$ are rational terms, then so are $t_1 + t_2$ and $t_1 - t_2$.
4. If $t$ is a rational term, then so is $-t$.
5. If $c$ is a rational number and $t$ is a rational term, then $c * t$, $t * c$ and $t/c$ are rational terms.

#### Constraints on linear rational terms

Given two rational terms $X$ and $Y$ the following constraints,

$$X > Y,\ X \geq Y,\ X < Y,\ X \leq Y,\ X = Y,\ X \neq Y$$

are all well-formed constraints of CHIP.

#### Constraint solving

CHIP provides a decision procedure for all constraints on linear rational terms. This means that for any set of constraints (over rational terms) CHIP can decide if the set of constraints has a solution. This decision procedure is an adaptation of the simplex algorithm based on variable elimination. Some desirable properties of this decision procedure are:

- Incrementality. Adding a new constraint to a solvable collection of constraints does not require entirely resolving the original collection. This is an important property because in CHIP constraints are created dynamically.
- Symbolic solution. The solution returned by the system is always the most general one.
- Optimization. CHIP can be used for deciding if a set of constraints is satisfiable and for finding the most general solution to a set of constraints which optimizes a linear evaluation function.

### 2.2.4 Control

Apart from the introduction of new computation domains, CHIP also provides control mechanisms such as demons and delay mechanisms. The delay mechanism is a generalization of Lee Naish's wait mechanisms. Delay mechanisms not only enable sound implementation of negation as failure, some higher order extensions and standard Prolog arithmetic, but are also a powerful tool for implementing local and conditional propagation techniques.

### Delay mechanism

Delay mechanisms are special control mechanisms. They postpone the 'proof' of a goal until its pre-conditions can be satisfied. Delayed goals are tried as soon as their pre-conditions are satisfied. Example:

```
delay not(X) if nonground(X).

not(X) :- call(X),!,fail.
not(X).
```

The goal `not(X)` delays if its argument is nonground (i.e. contains free variables). [2]

[2] By the way: This is a sound implementation of negation as failure.

# Chapter 3

# The ALM model

A prototype of an ALM model was developed, and implemented in CHIP. A description of the model is presented. It is shown how the ALM model can be implemented in CHIP. Some ALM modeling results are presented.

## 3.1 Formulation of the ALM model

### 3.1.1 Representation

#### Notation

$N_j$ denotes the net interest income in period j,
$V_{ij}$ the volume of section i of the balance sheet in period j, and
$R_{ij}$ the interest rate of section i in period j.

#### Net Interest Income

Net interest income $N_j$ is defined as

$$N_j = \sum_{i=1}^{A} V_{ij} R_{ij} - \sum_{i=A+1}^{L} V_{ij} R_{ij}$$

where $i = 1, \ldots, A$ are assets and $i = A+1, \ldots, L$ are liabilities.

#### Interest Rates

Every section of the balance sheet usually consists of subsections. The interest rate of a section is the weighted average of the interest rates of its subsections. Like [24], we assume that the adjustment of the interest rates can be described using a difference equation. This equation is known as the partial adjustment equation.

$$R_{ij} = (1 - \varphi_i) R_{ij-1} + \varphi_i R^p_{ij}$$

$R^p_{ij}$ denotes the interest rate of the new production. The coefficient $\varphi_i$ can be estimated using linear regression. The interest rate of the new production is a result of the market mechanism. We estimate this interest rate using an interest indicator $R_{mj}$. ($\alpha_i$ and $\beta_i$ are coefficients)

$$R^p_{ij} = \alpha_i R_{mj} + \beta_i$$

### Volumes

To describe the volumes of the several sections we use a slightly different approach. The volume of a section can increase in two ways. The bank can increase $V_{ij}$ by increasing its market-share $S_{ij}$, or $V_{ij}$ can increase because the total market $T_{ij}$ of section i increases (ceteris paribus). We assume that $T_{ij}$ is independent.

For some sections i, $V_{ij}$ has a lower bound (other than 0) because not all money is paid back at once. The fraction of $V_{ij-1}$ that is paid back in period j is denoted by $\mu_i$.

The bank can increase its market-share in two ways. The first one has no (direct) effect on $R^p_{ij}$, for example, advertisement, opening of more sales offices, et cetera. The second way is the price weapon. In this case the bank will try to control the volume of section i by changing $R^p_{ij}$. For some assets or liabilities the bank will not try to increase its market share. These assets and liabilities are so wanted by all the banks that increasing (decreasing) $R^p_{ij}$ will have no influence on $V_{ij}$ because the competition will react immediately.

If $T_{ij} = (1 + g_{ij})T_{ij-1}$ and $S_{ij} = (1 + \tau_{ij})S_{ij-1}$ then

$$V_{ij} = (1 + \tau_{ij})(1 + g_{ij})V_{ij-1}$$

where $Lb_i \leq \tau_{ij} \leq Ub_i$ and

$$V_{ij} \geq (1 - \mu_i)V_{ij-1}$$

$Lb_i$, $Ub_i$, and $g_{ij}$ are independent model parameters ('input data').

### Validity of Assumptions

Loans can be paid back in three ways: annuity, lump-sum, linear. In our model we assume that $\mu_i$ and $\varphi_i$ are constant. In reality $\mu_i$ and $\varphi_i$ will vary between boundaries. Let $P_{ij}$ denote the new production of balance sheet item $i$ in period $j$. If we assume that $V_{ij} = (1 - \mu_{ij})V_{ij-1} + P_{ij}$ and $(1 + L)P_{ij-1} \leq P_{ij} \leq (1 + U)P_{ij-1}$, we can derive that, $\forall_{j>D}$,

$$(1 + L)V_{ij} \leq V_{ij+1} \leq (1 + U)V_{ij}$$

and,

$$\frac{1}{\sum_{t=0}^{D-1}(1+U)^t} \leq \mu_{ij} \leq \frac{1}{\sum_{t=0}^{D-1}(1+L)^t}$$

when payments are lump-sum and,

$$\frac{\sum_{t=0}^{D-1}(1+U)^t}{\sum_{t=0}^{D-1}(1+t)(1+U)^t} \leq \mu_{ij} \leq \frac{\sum_{t=0}^{D-1}(1+L)^t}{\sum_{t=0}^{D-1}(1+t)(1+L)^t}$$

when payments are linear. D denotes the term of the loan. So given L and U we can compute an upper and a lower bound of $\mu_{ij}$ when payments are linear or lump-sum. If we also assume that $R^p_{ij} = (1 + R)R^p_{ij-1}$ we can derive that, $\forall_{j>D}$,

$$\frac{1}{\sum_{t=0}^{D-1}(1+U)^t(1+R)^t} \leq \varphi_{ij} \leq \frac{1}{\sum_{t=0}^{D-1}(1+L)^t(1+R)^t}$$

when payments are lump-sum and,

$$\frac{\sum_{t=0}^{D-1}(1+U)^t(1+R)^t}{\sum_{t=0}^{D-1}(1+t)(1+U)^t(1+R)^t} \leq \varphi_{ij} \leq \frac{\sum_{t=0}^{D-1}(1+L)^t(1+R)^t}{\sum_{t=0}^{D-1}(1+t)(1+L)^t(1+R)^t}$$

when payments are linear. Given L, U and R we can compute an upper bound and a lower bound for $\varphi_{ij}$.

This analysis shows that some sections can violate the partial adjustment assumption. This may cause incorrect model results. To prevent serious errors these sections will have to be modeled in the orthodox way. Especially when annuities are used to calculate the payments this can be very hard.

### 3.1.2 Asset & Liability Constraints

#### Stability

The stability constraint prevents sharp decreases in NII. The NII in period $j$ should not decrease more than a fraction $s$ of NII in period $j-1$.

$$N_j \geq sN_{j-1}$$

The best thing would be to have $s \geq 1$. Unfortunately this is not always possible. Finding the right value for $s$ can be troublesome. It is interesting to know the maximum value of $s$ given a certain interest rate scenario because it can serve as a measure for risk. Later on it will be explained how this maximum value can be found without too much programming effort.

#### Margin

Because most of the spreads are positive the bank can always increase its NII by increasing the total amount of assets and liabilities. However, this increase in NII may be very expensive and result in a lower net income. The margin constraint states that the margin (i.e. NII divided by total assets) must always be more than a certain minimum $m$.

$$N_j \geq m \sum_{i=1}^{A} V_{ij}$$

where $i = 1, \ldots, A$ are assets.

#### Capital Ratio

The capital ratio is defined as equity divided by total liabilities. The capital ratio constraint is implemented in a special way. The capital ratio must always be equal to the capital ratio of the starting balance (i.e. the capital ratio in period 0).

#### Liquidity and Solvability

The liquidity constraint is used to prevent liquidity traps. There are two sets of liquidity and solvability constraints. The Dutch Central Bank (DNB), supervises the liquidity and solvability positions of the banks. The bank itself also defines rules for liquidity and solvability. A liquidity constraint is of the form

$$\sum_{i=1}^{A} r_i V_{ij} \geq \sum_{i=A+1}^{L} r_i V_{ij}$$

where $i = 1, \ldots, A$ are assets, $i = A+1, \ldots, L$ are liabilities, and $0 \leq r_i \leq 1$. The solvability constraint is of the form

$$V_{ej} \geq \sum_{i=1}^{A} f_i V_{ij}$$

where $V_{ej}$ denotes the total amount of equity, $i = 1, \ldots, A$ are assets, and $0 \leq f_i \leq 1$. The liquidity constraint states that for every short term liability a certain amount of short term or marketable assets must be present. The solvability constraint states that the amount of equity must always be more than the summation of fractions of the asset volumes.

**Total Assets = Total Liabilities**

The balance sheet is always balanced. Therefore,

$$\sum_{i=1}^{A} V_{ij} = \sum_{i=A+1}^{L} V_{ij}$$

where $i = 1, \ldots, A$ are assets and $i = A+1, \ldots, L$ are liabilities.

### 3.1.3 Extra Rules

Every balance sheet section usually consists of subsections. Due to changes in the economic environment there may be a shift from one subsection to the other. These movements can affect the correctness of the model. To prevent this the model can use special rules to compute the effects of these movements. The prototype model contains two examples.

#### Movements in deposits

We can divide the section 'deposits' in several subsections with different terms. Let's suppose there are deposits with a one month term and deposits with a three months term. If investors expect an increase in short term interest rates they will invest their money in one month deposits. This means that the rate of adjustment of the interest rates $\varphi$ of the section 'deposits' increases. If investors expect a decrease in short term rates they will invest in three month deposits. This means that $\varphi$ will decrease. In our model we assume that the investors have perfect foresight (expectations theory) and therefore a significant (e.g. 0.5 percent) increase in short term rates in period $j$ compared to period $j-1$ means that $\varphi$ is higher than normal and a significant decrease means that $\varphi$ is lower than normal.

#### Movements in mortgage loans

Long term mortgage loans (e.g. 15 years) are usually more expensive than shorter term mortgage loans (e.g. 7 years). When the interest rates are very high there is shift from long term mortgage loans to shorter term mortgage loans.

### 3.1.4 Securities

Securities are a special balance sheet item because they are marketable. In our model we have made the following assumptions.

- If the bank sells securities it will always sell a combination of securities which has the same average interest rate and term as the balance sheet item 'securities'.
- If the bank sells securities in period $j$ then it will not buy securities in period $j$.
- The bank will not sell all its securities in the same period.
- The profit or loss generated by selling securites will be spread across a number of periods. Suppose the average term of securities is $n$ periods, then $1/n$ part of the profit or loss will be added to the NII of $n$ periods starting in the present period.
- Some of the profit or loss generated by selling securities will be accounted to periods outside the simulation period. The summation of these values must be positive.

The actual profit generated by selling securities is an effect of the market mechanism. We estimate this profit $P_{ij}$ using the following equation. Let $\Delta V_{ij}$ denote the amount of section $i$ that is sold in period $j$.

$$P_{ij} = n\Delta V_{ij}(R_{ij} - R^p_{ij})$$

where

$$0 \leq \Delta V_{ij} \leq (1 - \mu_i)V_{ij-1} - (1 - l_i)V_{ij-1}$$

$i$ = 'securities', and $(1 - l_i)V_{ij-1}$ is the lower bound of $V_{ij}$.

### 3.1.5 Goal Function?

The aim of ALM is to find a high and constant NII subject to several constraints (for example liquidity and solvability). In classical linear programming models there is usually one goal function: to maximise profit. In our model there is more than one simple goal. Moreover, the priorities of the goals may change depending on the results of the model. In other words, the bank can consider different strategies or combinations of strategies.

#### Greedy

The greedy strategy tries to maximize $N_j$ in every period. First it maximizes $N_1$, then it maximizes $N_2$, et cetera. This is an interesting strategy because to some extent it simulates the behavior of a bank without special AL regulations. Because the bank does not anticipate or plan, the movements in NII will be large (the stability is low). Moreover, this strategy probably leads to a lower total net interest income.

#### Maximize the stability

Instead of using the stability as a constraint we can use it as a goal. By using an iterative method we try to maximize the minimum stability. The result of this method is that,

$$\forall_j, \quad N_j \geq sN_{j-1}$$

and

$$\exists_j, \quad N_j = sN_{j-1}$$

where $s$ denotes the minimum stability.

#### Maximize the margin

This strategy tries to maximize the minimum margin. The result of this method is that,

$$\forall_j, \quad N_j \geq m\sum_{i=1}^{A} V_{ij}$$

and,

$$\exists_j, \quad N_j = m\sum_{i=1}^{A} V_{ij}$$

where $i = 1, \ldots, A$ are assets, and $m$ denotes the minimum margin.

#### Combinations

It is of course also possible to try combinations of these strategies. In such a case we have a multiple objectives model with absolute priorities on the goals. Example: `maximize the margin` followed by `maximize the stability`, is not the same as, `maximize the stability` followed by `maximize the margin`.

## 3.2 Implementation of the prototype ALM model

Most of the implementation is quite straightforward for someone familiar with Prolog or logic programming. A (modified) listing of the program is included in the appendix. We only describe essential parts of the model. First we describe the most basic version. Then we show how some of the extensions can be implemented.

**Notation.** To be able to distinguish predicates on rational terms and the normal arithmetic predicates, the 'rational' predicates are preceded by the **$** sign.

### 3.2.1 Basic Model

The model consists of four parts: translation of input data, creating the constraints, optimization, and translation of output. The top level of the program looks like this:

```
top(Scenario,Time,Total) :-
    format_starting_balance(Nii,Volumes,Rates),
    generate_constraints(Scenario,0,Time,Total,Nii,
                         Volumes,Rates,Results),
    rmax(Total),
    output_results(Results).
```

The program does the following: it retrieves information about the starting balance from the (Prolog) database, then it creates all the ALM constraints, maximizes total net interest income and finally outputs the results in a nice format, e.g. business graphics.

The constraints state relations which must hold between variables in the same period and relations which must hold between the variables in two adjacent periods. The only thing we have to do is to describe the general pattern and then tell CHIP to repeat this pattern. In other words, the constraints can be created using a recursive loop. Let's have a look at a small example.

```
top :- create_constraints(0,20,10).

create_constraints(Time,Time,_).
create_constraints(Time0,Time2,V0) :-
     Time0 < Time2,
     Time1 is Time0 + 1,
     V1 $=< 1.1 * V0,
     create_constraints(Time1,Time2,V1).
```

This program creates all the constraints which are described by the following formula.

$$\forall_{0<j<21},\ V_j \leq 1.1 * V_{j-1},\quad V_0 = 10$$

The `generate_constraints` predicate in the ALM model is implemented using a recursive loop.

The predicate `rmax/1` is a built-in feature of CHIP. It maximizes the value of its argument. This predicate is a higher-order predicate. We must always be very careful when using higher-order predicates or extra-logical extensions in CHIP and in Prolog. In fact, problems with higher-order predicates can occur in any programming language. Example:

```
p1(A) :-
   A $=< 3,
   rmax(A).
p2(A) :-
   rmax(A),
   A $=< 3.
```

The two clauses do not express the same 'relation'! The goal `:- p2(A)` results in an error message [1], whereas `:- p1(A)` gives the correct answer: 3. Such a behavior cannot be avoided when using higher-order predicates. It would be more correct to write the maximization predicate in the following way.

```
rmax(Function, Constraints) :-
    call(Constraints),
    maximize(Function).
```

However, the 'strange' behavior of maximizations is also desirable. CHIP creates constraints dynamically and always returns the most general solution. This also happens when `rmax/1` is used. Let's have a look at a small example.

```
p1(A) :-
   A $>= 0,
   A $=< 1,
   B $>= 2,
   B $=< 3,
   C $= B - A,
   rmax(C),
   rmax(A).
```

```
p2(A) :-
   A $>= 0,
   A $=< 1,
   B $>= 2,
   B $=< 3,
   C $= B - A,
   rmax(A),
   rmax(C).
```

The result of `:- p1(A)` is `A = 0` and the result of `:- p2(A)` is `A = 1`. In operations research terminology this program is called a multiobjective linear programming model with absolute priorities on the goals.

### 3.2.2 Maximization

The implementation of the maximization strategies is straightforward, because CHIP always returns the most general solution [2]. Example:

```
/* [Nii1|Rest] is a list of NIIs: [Nii1,Nii2,...,Niin] */

greedy_maximization([Nii1|Rest]) :-
   rmax(Nii1),
   greedy_maximization(Rest).
```

Maximization of the stability or the margin is trickier because in such a case the objective function is non-linear.

The stability constraint was defined as follows:

$$N_j \geq sN_{j-1}$$

We can use an iterative method to find the maximum value of $s$. There are several ways to program this. The following program uses a bisection [3] method. The predicates `setval/2` [4] and `getval/2` are used to simulate global variables. The predicate `maximize_stability` is called after `generate_constraints` and before the maximization, e.g. `greedy_maximization`.

```
/* List = [Nii0,Nii1,...,Niin] */

maximize_stability(List,Lowerbound,Upperbound,Precision) :-
```

[1] Remember the try order is from left to right.

[2] That is, the rational arithmetic part of CHIP

[3] Bisection is very slow but also general and stable.

[4] `setval/2` is a built-in of the CHIP compiler it works like `setq` in LISP.

```
    setval(temp,[Lowerbound,Upperbound]),
    repeat,
        getval(temp,[L0,U0]),
        Stability $= (L0 + U0) / 2,
        (testval(List,Stability) ->
            L1 = Stability, U1 = U0;
            L1 = L0, U1 = Stability),
        setval(temp,[L1,U1]),
        U0 - L0 $=< Precision,
    !.

testval([_],_).
testval([Nii0,Nii1|Rest],Stability) :-
    Nii1 $>= Stability * Nii0,
    testval([Nii1|Rest],Stability).

repeat.
repeat :- repeat.
```

Members of the religious declarative programming movement will be deeply shocked on seeing this program. It is a good example of non-declarative programming. It could have been implemented recursively without using global variables. [5] The advantages of this implementation are that it uses less memory space and is more efficient.

### 3.2.3 Securities

The implementation of the securities part is rather tricky. The bank can either sell or buy securities. If the bank buys securities it does not sell securities and vice versa. This means that there is a disjunctive constraint. If we simulate 20 time steps, i.e. periods, there are $2^{20}$ possibilities. Explicit enumeration of all the possibilities is impossible. Implicit enumeration methods like branch and bound require a good estimation function. In our case there is no good estimation function. Moreover, translation of the model to a mixed integer programming model is not straightforward.

Selling securities has two effects. The direct effect is the profit generated by selling them. The indirect effect is the effect on NII. Determining the total effect requires maximization of the objective function.

In the present implementation we use explicit enumeration to find the 'optimal solution'. If we simulate 20 periods we just limit the number of periods in which securities can be sold. This solution is not satisfactory.

There is, however, a heuristic which can help us to find a good solution. First we maximize without selling securities. This gives an initial solution. Then we try to find the periods where the volume of securities is very near its lower bound (i.e. where $V_j \approx (1-\mu)V_{j-1}$). We try to improve the initial solution by selling securities in these periods.

### 3.2.4 Translation of output

One of the advantages of CHIP is that it allows processing of the results. CHIP provides a number of predicates to produce graphical output. The prototype ALM model displays the results of the optimization in the form of business graphics.

[5] This is a strange implementation indeed. The problem is that we only want to create 'the most constraining constraint'. All the other constraints are redundant.

### 3.2.5 Model development in CHIP

Several versions of the ALM model have been developed using CHIP. This was done to test new ideas. What is most striking is that development times were very short. No version of the ALM model took more than a week to complete. Implementing the same type of model in Pascal or C would have taken several months. Modifying the programs is easy as well. The programs are short, modular and easy to read.

The next section presents some results generated by the ALM prototype model. Very often it was necessary to change small parts of the program to be able to run these experiments. These modifications usually took very little time.

The prototype ALM model does not have a very sophisticated user-interface. The CHIP compiler provides facilities to build spreadsheet-like interfaces. These were not yet available at the time the prototype ALM model was being developed and therefore the user-interface of the prototype is a bit primitive.

## 3.3 ALM model results

The ALM model has been run many times using different sets of data and different assumptions. This was done to test the model and to develop new ideas. [6] The examples illustrate the necessity of sensitivity analysis.

### 3.3.1 Model description

The prototype ALM model describes six assets and five liabilities (including equity). One simulation step, i.e. period, is equal to three months. So when we simulate twenty time steps we simulate five years.

#### Balance sheet items

The following table lists the several balance sheet items.

| Item number | Name |
|---|---|
| 1 | Short term loans |
| 2 | Current account (overdraft) |
| 3 | Consumer loans |
| 4 | Mortgage loans |
| 5 | Long term loans |
| 6 | Securities and private placements (marketable) |
| 7 | Short term time deposits |
| 8 | Current account |
| 9 | Savings |
| 10 | Long term funding |
| 11 | Equity |

### 3.3.2 Scenarios

The model has been run using different interest rate scenarios. Because little is known about future interest rates, we use several alternative scenarios. By comparing the results of different scenarios the model-user tries to assess the present income and risk positions and to find possible strategies to alter the current exposure to the desired exposure.

An interest scenario describes the time-dependent values of the interest rate indicators. [7]

### 3.3.3 Graphics

The results of the optimization are presented in the form of business graphics. Figure 3.1 is an example. The graph in the lower left corner is the interest rate scenario. In this particular example the short term rates increase. This results in a totally flat yield curve. The graph in the upper left corner shows the effect of this increase on NII. The NII in period eight decreases. The seven small graphs on the right display two things. The dotted lines are the time-dependent average interest rates of the balance sheet (BS) items. The minimum and maximum values are written on the right side of the boxes. The solid lines display the time-dependent volumes of the BS-items which maximize the goal function(s). The minimum

[6] The ALM model was tested using real-life data from the Amro Bank. The data used in the following examples, however, is fictitious.

[7] In this model we use the following indicators; AIBOR: Amsterdam Inter Bank Offered Rates (3 months); K: The yield on government bonds with time to maturity 5-8 years; PD: The banking rate or discount rate; PDplus: The banking rate with a special premium; AK2: The average of AIBOR and K.

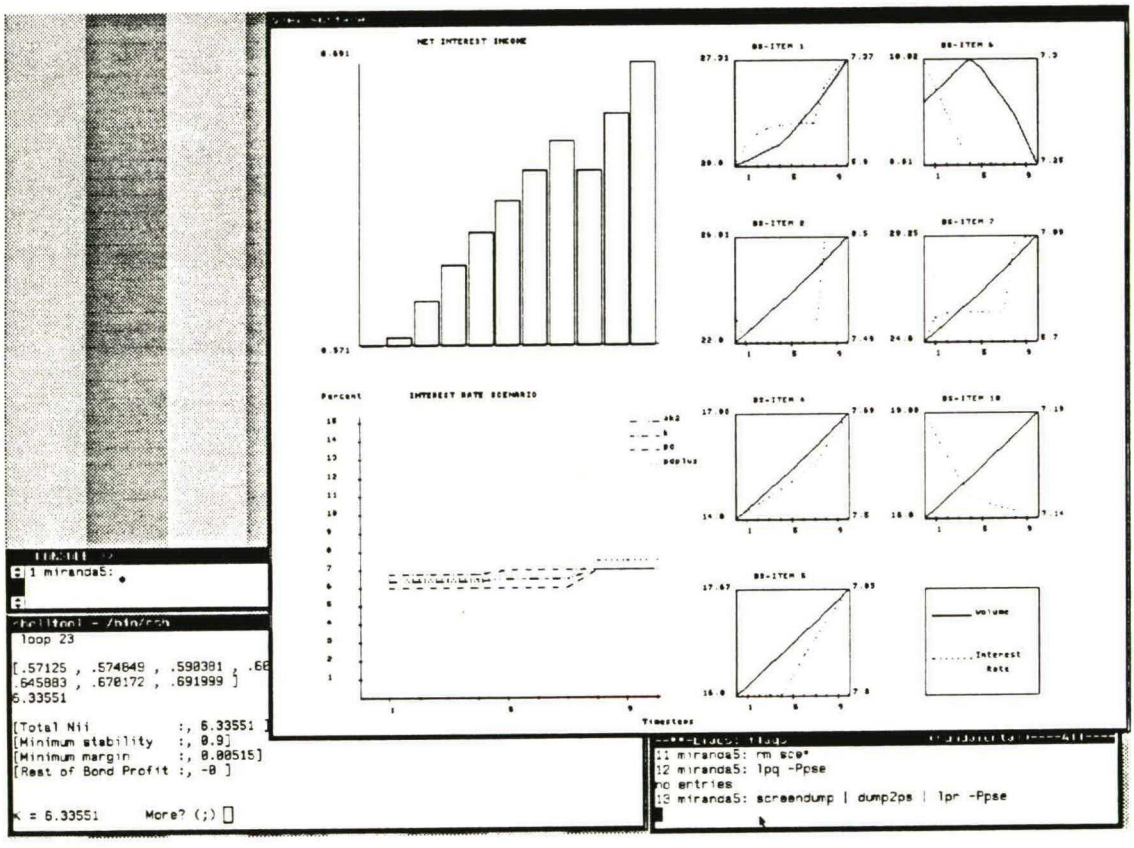



**Figure 3.1: Maximization of total NII; scenario 1**

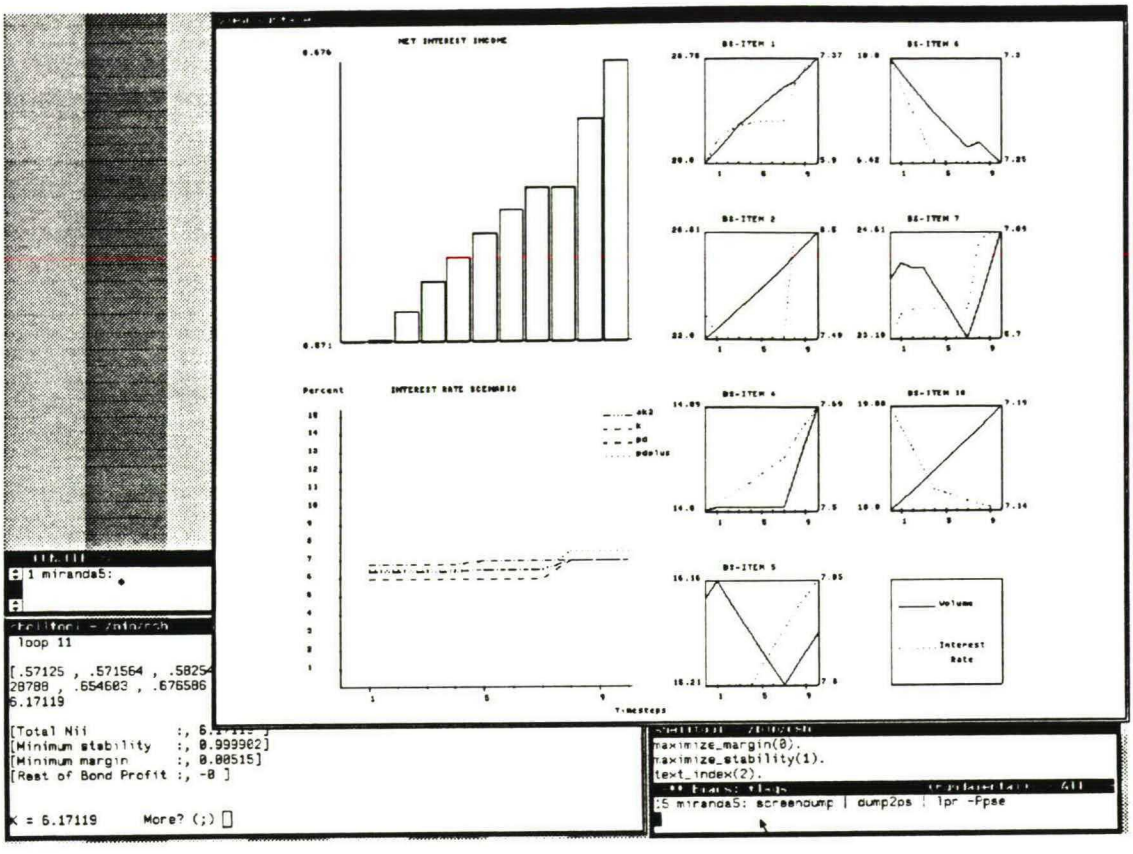



Figure 3.2: Maximization of stability and total NII; scenario 1

and maximum values are written down on the left side of the boxes. The BS-items that are displayed are BS-items 1, 2, 4, 5, 6, 7, and 10. BS-items 3, 8, 9, and 11 are not displayed. We assume that BS-items 3, 8, and 9 are very profitable and therefore the bank always tries to increase these items. BS-item 11, equity, is a fixed percentage of the total assets (or liabilities) and therefore not really independent. The small window beneath the graphics window provides additional information: minimum stability, minimum margin, the profit generated by selling securities which is accounted to periods after the last simulation step ('rest of bond profit'), and total NII.

### 3.3.4 Examples

Figure 3.1 is a screen-dump taken from the system. It shows the results of the model when maximizing the total NII. The stability and margin were used as normal constraints. We can see that there is a small decrease in NII in period eight. Figure 3.2 shows the results when we first maximize the stability and then maximize the total NII. There are quite a few differences with the previous picture. The dip in NII in period eight is gone. Unfortunately, this 'stability' comes at a high price; the total NII has decreased significantly. The differences in the volumes of the BS-items are interesting as well. Consider for example BS-item four, i.e. mortgage loans. In the first example the volume of mortgage loans is increasing. In the second example the volume of mortgage loans is constant in the first seven periods. If we only want to maximize NII, increasing the number of mortgage loans is perhaps a good idea. If, however, the stability is important as well then we must keep 'mortgage loans' constant.

The dip in NII in period eight is rather small. Maximizing the stability resulted in a

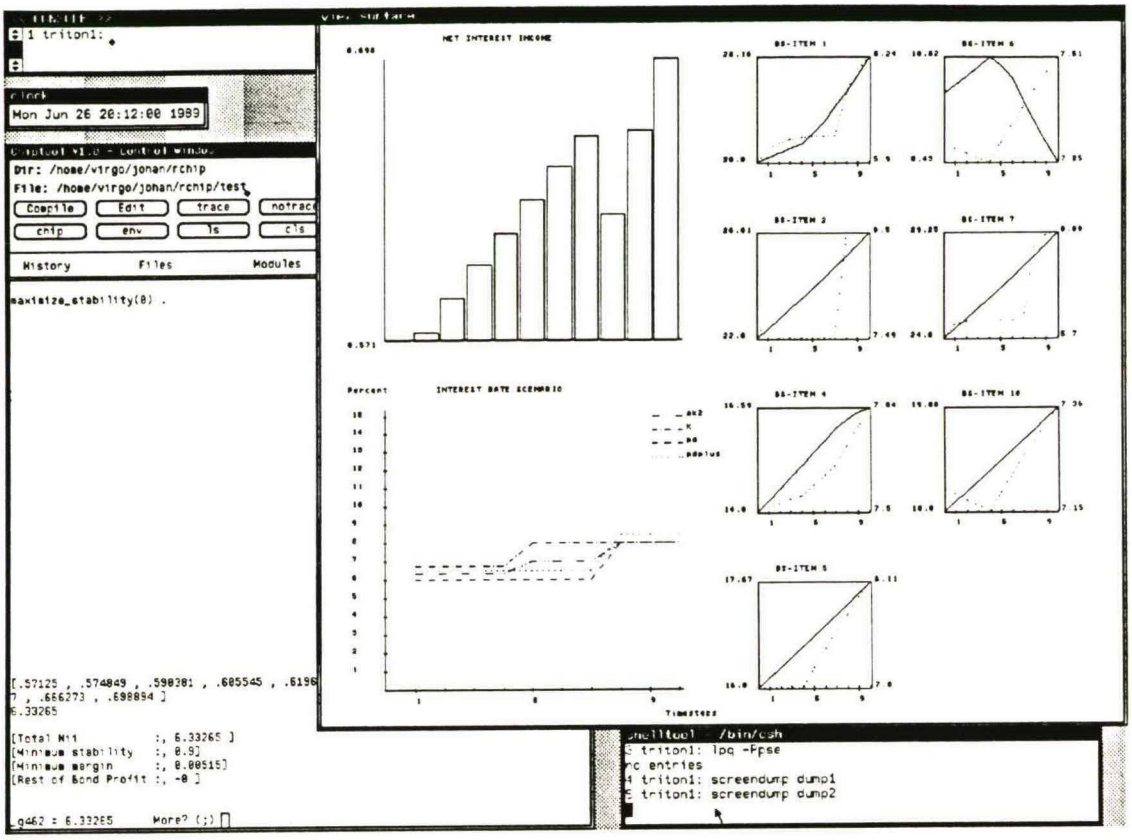



Figure 3.3: Maximization of total NII; scenario 2

significantly lower total NII. We could conclude the increasing the volume of mortgage loans is a good idea. However consider the next example 3.3. In this example we use a different interest rate scenario. The direction of change of the interest rates is the same but the magnitude is higher. The dip in NII in period eight is now much more pronounced. This example clearly illustrates the tradeoff between risk and income.

### 3.3.5 What's best?

Interpreting the results of the simulation can be difficult. A balance sheet strategy which results in a good performance using scenario 'A' may cause a bad performance using scenario 'B'. It is therefore not easy to say which strategy is the best one. The only thing we can do is trying to make the advantages and disadvantages explicit by simulating the effects of different interest rate scenarios and balance sheet strategies.

# Chapter 4

# Conclusions

## 4.1 ALM modeling

We have seen that the ALM model allows us to answer so-called 'what-if' questions and to perform all kinds of optimizations. The results of the simulation are presented in the form of business graphics. This helps the model-user to interact with the system. In the following paragraphs we present some conclusions about ALM modeling and the ALM prototype.

### 4.1.1 Multiple objectives

The definition of the aim of ALM implies that the bank has more than one simple goal. The bank wants to maximize NII but it also wants to avoid 'risky situations'. In other words, the ALM problem is, like most real-life problems, a multiple objectives problem. Moreover, there is an absolute priority on the goals. The goal priorities are not constant in time, but may be influenced by changes in the internal or external environment of the bank.

### 4.1.2 Non-linearity

The ALM problem is, by nature, a 'non-linear problem'. Translating the problem to a linear equalities and inequalities model causes loss of information. The prototype ALM model, however, allows maximization of (simple) non-linear objective functions.

### 4.1.3 Sensitivity

CHIP does not generate the standard 'sensitivity analysis' like that in simplex packages. It is, of course, possible to program something like that in CHIP. However, we argue that the value of this kind of sensitivity analysis is limited. The balance sheet is always balanced. As a result it is impossible to change the volume of exactly one BS-item. We are more interested in the influence of combinations of changes. The prototype ALM model provides several methods to structure such simulations.

### 4.1.4 Disjunctive constraints

Several assumptions have been made about the trading of securities. One of the assumptions was that if the bank sells securities in period $j$ then it will not buy securities in period $j$. We have made this assumption because we use a difference equation, the partial adjustment equation, to estimate the average interest rate of the BS-item 'securities'. Unfortunately, 'security swaps' cannot be modeled without violating the partial adjustment assumption.

This can be seen by considering the formulas derived in section 3.1.1. Selling and buying securities at the same time, i. e. 'security swaps', causes a high variability of the partial adjustment parameter $\varphi$, whereas we assume that this parameter is a constant. We therefore assume that the bank does not sell and buy securities in the same period. Consequently, the trading of securities is a problem that involves disjunctive constraints. Although, it is possible to represent these disjunctive constraints in CHIP, we have found no general and efficient method to solve them. Formulation of the problem as a 'branch and bound' model is not straightforward.

### 4.1.5 Dynamic modeling

Simulation models are dynamic models. The maturity gap and the duration gap approach are both static models. The experiments with the ALM model have shown that changing the number of simulation steps sometimes results in a completely different solution. Of course, intuitively we already expected this; sometimes it is possible to compensate an initial loss with profits later on and vice versa. Simulation models help us to recognize these special situations. The static maturity gap and duration gap approach do not. Considering dynamic duration gap models might therefore be a good idea.

### 4.1.6 Precision

We have assumed that the BS-items can be described using an adjustment equation. This very simple representation is very flexible. We can for instance study the effects of changes in the adjustment parameters. Unfortunately this representation is not very precise. The results of our prototype ALM model have to be checked using a model with a more precise representation.

### 4.1.7 Operationality

The ALM prototype model has not yet been used in an actual ALM working environment. This is because the CHIP compiler is still under development. We can therefore conclude little about the practical industrial value of the prototype.

However, we have confidence in the value of the model and the CHIP approach. The results that have been presented to the ALM experts seemed to correspond with their intuitions. The development times of the model and the flexibility of the model make our approach very promising.

## 4.2 Evaluation of CHIP

CHIP provides three new computation domains: finite domains, Booleans and rational arithmetic. This section presents an evaluation of the rational arithmetic part of CHIP as an ALM modeling language.

### 4.2.1 Advantages

CHIP is, like LISP and Prolog, a symbolic programming language. Some of the advantages of CHIP, are in fact advantages of this class of programming languages. These languages enable symbolic computation, stratified design and are highly interactive.

The main features of the CHIP and Prolog are the relational form, unification, and non-deterministic computation. In addition CHIP provides sophisticated and efficient constraint solving techniques and control mechanisms. The constraints are created dynamically and the constraint solver is incremental.

#### Declarative programming

One of the main advantages of CHIP is that it enables a declarative style of programming. If you know that the relations you have stated are correct and actually express what you want CHIP will give you the right [1] answer.

#### Easy to read

The constraints state relations between objects, directly in the intended domain of discourse. This makes programs relatively easy to read. We do not have to simulate a constraint solver in our heads to understand what is written.

#### Concise representation

The set of constraints that describes the problem is at the same time an implicit representation of the solution. The constraints do not explicitly state how CHIP must derive the answer. This makes programs very short.

#### Top-down design

CHIP supports, in a natural way, top-down design. Indeed, in (constraint) logic programming, problems are solved top-down. The goal clause consists of 'subgoals' which in turn can consist of subgoals. To solve the problem we must solve all the subproblems.

#### Easy to modify

CHIP programs are easy to modify. This advantage is actually a consequence of the other advantages. CHIP programs are short and easy to read. CHIP supports declarative programming and top-down design. The programs can be structured and modular.

#### Flexibility

CHIP enables us to express constraints of different types in one program. The constraint solvers are fully embedded in a high-level programming language.

### 4.2.2 Disadvantages

#### Memory space management

In LISP, Prolog, and CHIP, memory space is allocated dynamically. On the positive side this makes the language very flexible. The negative effect is that if the space is not retrieved when no longer necessary the program will use a lot of memory space. Most of the LISP implementations employ a special program, called 'garbage collector', to retrieve memory space that is no longer needed. Garbage collection in LISP is well understood and is no longer an issue. Garbage collection in Prolog, on the other hand, is not in the same state of development as garbage collection in LISP. Although good theories exist about garbage collection in Prolog there are, currently, only few systems which actually have a reasonable garbage collector. Howevr, garbage collection in Prolog is a problem for which good solutions exist and is merely an implementation issue.

CHIP does not provide a garbage collector. The experiments with the ALM model have shown that at least some partial form of garbage collection is necessary.

[1] See ?? for possible exceptions

**Inefficiency**

The old LISP implementations were infamous for their inefficiency. Although LISP will never be as efficient as imperative languages on standard hardware, much improvement has been made. Currently there are many LISP implementations with a more than acceptable performance. The same story is true for Prolog. Up-to-date Prolog implementations perform at an acceptable level.

So what about the performance of CHIP? This question is difficult to answer. The question of course is: Compared to what?

CHIP provides a unique combination of techniques. Comparing CHIP with classical programming languages is unfair because these languages do not provide the same functionality. Comparing CHIP with specialized constraint solvers written in a procedural language is also unfair, because these programs usually do not provide the same flexibility. Moreover, we must keep in mind that CHIP and constraint logic programming are very recent developments, whereas most of the other tools have a long development history.

Several CHIP examples using the finite domains and Booleans have shown that an efficiency comparable to that of specialized constraint solvers (written in procedural languages) is possible [9, 12, 13]. In some cases CHIP even outperformed these specialized systems [29].

At the moment this is not the case when using the rational terms. Recent tests have shown that efficient execution of rational arithmetic in CHIP is achievable. It is, however, unlikely that an efficiency comparable to that of specialized simplex packages will be reached. This is due to the fact that constraints are created dynamically in CHIP. Moreover, the constraints are created in a non-deterministic environment.

## 4.3 Prototyping in CHIP

The ALM model is the first real-life financial application of the rational arithmetic part of CHIP. In fact, to my knowledge, it is the first real-life financial application of the rational/real arithmetic in constraint logic programming. All the other applications were taken from books and had already a well defined mathematical representation. In CLP(R), for instance, an option-trading program is presented. It uses the binomial and the Black-Scholes option pricing theory. Both theories are well defined and have been developed after many years of empirical research. In Prolog III an example of a small banking calculation is presented. Although this application is certainly elegant, it is too small to draw conclusions about the feasibility of constraint logic programming in the financial domain.

The ALM model was defined starting from scratch. There are many examples of ALM simulation models, ranging from specialized calculators to sophisticated mathematical programming or econometric models. There is, however, little consensus about what an ALM model should do. In fact, there is little consensus about the ALM theory.

There is more than one way to use CHIP for financial applications. The first way is to use CHIP as an application language. A well defined mathematical model exists and is represented in CHIP. The resulting program is the final product.

The second way is to use CHIP as a prototyping language. The problem does not have a well defined mathematical representation and is ill-understood. Successive prototypes are developed to be able to understand more about the problem and to sharpen the model specifications. Once there is a prototype which is satisfactory there are two possibilities.

- The prototype is used as the actual system.
- The prototype is thrown away and the actual system is build using the specifications of the prototype. The implementation language can be a different programming language like C or Pascal. The program can possibly use mathematical packages like simplex.

In our case, the prototyping approach was most appropriate. CHIP is an excellent language to support such an approach.

## 4.4 Do you really need it?

One could ask: What's new? Of course, models which use linear programming and mixed integer programming already existed before the invention (or discovery) of CHIP. What is new to the approach is that the constraint solvers are integrated parts of a high-level programming language with deductive capabilities.

Recent publications on Decision Support Systems and Computer Assisted Decision Making stress the need for integration of databases, modeling tools and graphics; see e.g. [4]. The constraint logic programming paradigm presents an overall framework for this integration.

# Appendix A

# Listing of the prototype ALM model

```
/*****************************************************************
*  Asset and Liability model in CHIP                             *
*  Version: Compiler version                                     *
*  Author : Johan M. Broek                                       *
*  Date   : 18/06/89                                             *
*                                                                *
*                                                                *
*                                                                *
*                                                                *
*****************************************************************/

/*****************************************************************
WARNING:

This is not a listing of the actual ALM model but a modified version.
The CHIP compiler is still being developed. Some nasty bits of
program, necessary to get the program running on the testversion of
the compiler, have been changed. Some other changes have been made to
improve clarity.

*****************************************************************/

:- cprolog.
:- dynamic stability/1, margin/1.

/*****************************************************************
*                       Toplevel                                 *
*                                                                *
*   S     = Identifier of the interest rate scenario            *
*   T     = The number of simulation timesteps                  *
*   Total = Total Nii                                           *
*                                                                *
*****************************************************************/

pr(S,T,Total) :-
```

```
   (grafics(yes) ->
       (open_cgi(0,0,2000,2000),
        writeln('Please resize window and type <return>'),
        get0(_),
        clear_view_surface(0));true),
    format_starting_balance(Nii,Volumes,Rates),
    generate_constraints(S,0,T,[],Securities,Nii,Volumes,Rates,Results),
    get_nii(Results,NiiList),
    sum_list(NiiList,Total),
    sum_list(Securities,SumSecurities),
    SumSecurities $>= 0,
    (maximize_stability(yes) ->
         create_stability_constraint([Nii|NiiList],1.0); true),
    rmax(Total),
    starting_balance(volumes(V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,Equity),
                     rates(R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,0)),
    Resultsa  = [results(0,Nii,
                     [V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,Equity],
                     [R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,0])|Results],
    (grafics(yes) ->
        (clear_view_surface(0),
        character_height(6),
        text_font_index(2),
        map_scenario(0,0,900,S,T),
        chart(0,1000,1000,minmaxbar,NiiList),
        map_behavior(1000,1500,500,1,Resultsa),
        map_behavior(1000,1000,500,2,Resultsa),
        map_behavior(1000,500,500,4,Resultsa),
        map_behavior(1000,0,500,5,Resultsa),
        map_behavior(1500,1500,500,6,Resultsa),
        map_behavior(1500,1000,500,7,Resultsa),
        map_behavior(1500,500,500,10,Resultsa));true).

/************************************************************
* Parameters and Flags                                      *
************************************************************/

/* flags */

grafics(no).
maximize_margin(no).
maximize_stability(no).

/*
Minimum margin
Margin = Net Interest Income / Total Assets
*/

margin(0.0052).

/*
Minimum stability for every period.
```

```
Nii(J) > Stability * Nii(J-1)
*/

stability(0.949).

/*
These are the parameters that describe the several balance sheet items.

P          = (ignore)
I          = Index number of BS-item
M          = Indentifier of the interest indicator
Phi        = The adjustmemt parameter of the average interest rate
             of the BS-item
Mu         = The fraction of the volume that is paid back in period
             J of BS-item I
Alpha,Beta = Parameters that the define the linear relationship
             between the production interest rate of I and its
             interest indicator
Ub, Lb     = Upper and Lower bound for the growth of the volumes

params(P,I, M,      Phi,    Mu,     Alpha,  Beta,  Ub,    Lb)
=============================================================
*/

params(P,1, aibor,  0.5,    -0.5,     1,     0.005,  0.05, -0.05).
params(P,2, pdplus, 1,      -1,       1,     0.01,   0.02, -0.02).
params(P,3, pd,     1,      -1,       1,     0.045,  0.02, -0.02).
params(P,4, ak2,    (1/28), (-1/100), 1,     0.015,  0.02,  0).
params(P,5, k,      (1/20), (-1/40),  1,     0.01,   0.01, -0.01).
params(P,6, k,      (1/20), -0.05,    1,     0.0025, 0.02, -0.08).
params(P,7, aibor,  0.8,    -0.8,     1,     0.001,  0.02, -0.01).
params(P,8, pd,     1,      -1,       0.5,  -0.001,  0.02, -0.01).
params(P,9, pd,     1,      -1,       0.8,  -0.004,  0.02, -0.01).
params(P,10,k,      (1/24), (-1/24),  1,     0.001,  0.01, -0.01).


/* The starting balance sheet. The last volume in the list
   is always equity. (p.s. The data is fictitious.)

*/

starting_balance(
  volumes(20.0, 22.0, 4.0, 14.0, 16.0, 10.0, 24.0, 16.0, 20.0, 18.0, 8.0),
  rates(0.059, 0.0775, 0.09, 0.075, 0.078, 0.073, 0.057, 0.029, 0.043, 0.072, 0)
               ).


/************************************************************/

format_starting_balance(Nii,Volumes,Rates) :-
    Volumes = volumes(V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,CapitalRatio),
    Rates = rates(R1,R2,R3,R4,R5,R6,R7,R8,R9,R10),
```

```
    starting_balance(volumes(V1,V2,V3,V4,V5,V6,V7,V8,V9,V10,Equity),
                     rates(R1,R2,R3,R4,R5,R6,R7,R8,R9,R10,0)),
    Nii $= (V1 * R1 + V2 * R2 + V3 * R3 + V4 * R4 + V5 * R5 +
            V6 * R6 -
            (V7 * R7 + V8 * R8 + V9 * R9 + V10 * R10)) / 4,
    CapitalRatio $= Equity / (V10 + V9 + V8 + V7 + Equity).


/*********************************************************************
*                   Generate Constraints                             *
*                                                                    *
*                                                                    *
*  S         = identifier of interest rate scenario                  *
*  T         = number of timesteps                                   *
*  Nii0      = Net Interest Income at time J - 1                     *
*  Se0       = List of security profits                              *
*********************************************************************/

generate_constraints(_,T,T,Se,Se,_,_,_,[]).
generate_constraints(S,J0,T,Se0,Se2,Nii0,
             Volumes,Rates,[Results1|Results2]) :-
    J0 < T,
    J is J0 + 1,
    Volumes = volumes(V10,V20,V30,V40,V50,V60,V70,V80,V90,V100,CR),
    Rates =  rates(R10,R20,R30,R40,R50,R60,R70,R80,R90,R100),
    NewVolumes = volumes(V11,V21,V31,V41,V51,V61,V71,V81,V91,V101,CR),
    NewRates = rates(R11,R21,R31,R41,R51,R61,R71,R81,R91,R101),
    Results1 = results(J,Nii2,
                [V11,V21,V31,V41,V51,V61,V71,V81,V91,V101,Equity],
                [R11,R21,R31,R41,R51,R61,R71,R81,R91,R101,0]),
    section(S,1, J,V10, V11, R10,R11),
    section(S,2, J,V20, V21, R20,R21),
    section(S,3, J,V30, V31, R30,R31),
    section(S,4, J,V40, V41, R40,R41),
    section(S,5, J,V50, V51, R50,R51),
    section(S,7, J,V70, V71, R70,R71),
    section(S,8, J,V80, V81, R80,R81),
    section(S,9, J,V90, V91, R90,R91),
    section(S,10,J,V100,V101 ,R100,R101),
    section(S,6, J,V60, V61, R60,R61,SL),
    add_two_security_lists(Se0,SL,SProfit),
    test_security_profit(SProfit,FirstSProfit,Se1),
    V11 + V21 + V31 + V41 + V51 + V61 $= TA,
/* Capital Ratio Constraint: Equity = (Equity0 / TA0) * TA */
    Equity $= CR * TA,
/* Total assets is equal to total liabilities */
    TA $= V71 + V81 + V91 + V101 + Equity,
/* Liquidity Constraint */
    V11 + 0.8 * V61 $>= 0.1 * V91 + 0.2 * V81,
/* Solvability Constraint */
    Equity $>= 0.04 * V11 + 0.04 * V21 +
               0.01 * V31 + 0.04 * V41 +
```

```
              0.04 * V51 + 0.01 * V61,
/* Definition of Net Interest Income */
    Nii1 $= FirstSProfit +
            (V11 * R11 + V21 * R21 + V31 * R31 +
             V41 * R41 + V51 * R51 + V61 * R61 -
            (V71 * R71 + V81 * R81 + V91 * R91 +
             V101 * R101)) / 4,
/* Policy constraints */
    V41 + V51 + V61 - V101 $=< 24,
    0.8 * (V81 + V91 + Equity) $>= V41 + V51 + V61 - V101,
    constrain_nii3(Nii1,TA),
    constrain_nii2(Nii0,Nii1),
/* Compute the constraints for the next period */
    generate_constraints(S,J,T,Se1,Se2,Nii1,
                         NewVolumes,NewRates,Results2).

/* Create Stability Constraint */

   constrain_nii2(Nii0,Nii1) :-
    stability(S),
    Nii1 $>= S * Nii0.

/* Create Margin Constraint */

constrain_nii3(Nii,TA) :-
    margin(M),
    Nii $>= M * TA.


/* Constrains the volume of a BS-item and calculates
   the interest rate.

S  = the identifier of the interest rate scenario
I  = the index number of the BS-item
J  = time
V0 = the volume of BS-item I at time J-1
V1 = the volume of BS-item I at time J
R0 = the average interest rate of BS-item I at time J-1
R1 = the average interest rate of BS-item I at time J
Rm = the value of the interest indicator at time J
*/

section(S,I,J,V0,V1,R0,R1) :-
   member(I,[1,2,3,4,5,8,9,10]),
   params(P,I,M,Phi,Mu,Alpha,Beta,Ub,Lb),
   rate(S,M,J,Rm),
   cmax(Mu,Lb,Max),
   V1 $=< (1 + Ub) * V0,
   V1 $>= (1 + Max) * V0,
   R1 $= (1 - Phi) * R0 + Phi * (Alpha * Rm + Beta).

/*
```

```
This BS-item (7) has a special computation rule for Phi.
If the interest indicator has increased significantly
(compared to J-1) then Phi is greater than normal.
If the interest indicator has decreased
significantly then Phi is smaller than normal.
*/


section(S,7,J,V0,V1,R0,R1) :-
   J0 is J - 1,
   params(P,7,M,Phi0,Mu,Alpha,Beta,Ub,Lb),
   cmax(Mu,Lb,Max),
   rate(S,M,J0,Rm0),
   rate(S,M,J,Rm),
   compute_phi(Rm0,Rm,Phi0,Phi),
   V1 $=< (1 + Ub) * V0,
   V1 $>= (1 + Max) * V0,
   R1 $= (1 - Phi) * R0 + Phi * (Alpha * Rm + Beta).

compute_phi(Rm0,Rm,Phi0,Phi) :-
     Rm  $> Rm0 + 0.005,
     Phi $= Phi0 + 0.1.
compute_phi(Rm0,Rm,Phi0,Phi) :-
     Rm  $< Rm0 - 0.005,
     Phi $= Phi0 - 0.1.
compute_phi(Rm0,Rm,Phi0,Phi) :-
     Rm  $>= Rm0 - 0.005,
     Rm  $=< Rm0 + 0.005,
     Phi $= Phi0.

/*
This is a special BS-item (6) because it is marketable
*/

section(S,6,J,V0,V1,R0,R1,List) :-
   params(P,6,M,Phi,Mu,Alpha,Beta,Ub,Lb),
   rate(S,M,J,Rm),
   V1 $=< (1 + Ub) * V0,
   Rp $= Alpha * Rm + Beta,
   R1 $= (1 - Phi) * R0 + Phi * Rp,
   buy_or_sell(J,V0,V1,Rp,R1,Mu,Lb,List).

/* Buy or sell securities */

buy_or_sell(J,V0,V1,Rp,R1,Mu,Lb,[]) :-
  J > 6,          % Dont sell securities after period 6
  cmax(Mu,Lb,Max),
  V1 $>= V0 * (1 + Max).
buy_or_sell(J,V0,V1,Rp,R1,Mu,Lb,[]) :-
  J < 7,
  Mu $=< Lb,
  V1 $>= V0 * (1 + Lb).
```

```
buy_or_sell(J,V0,V1,Rp,R1,Mu,Lb,[]) :-
  J < 7,
  Mu $> Lb,
  V1 $>= V0 * (1 + Mu).
buy_or_sell(J,V0,V1,Rp,R1,Mu,Lb,S) :-
  J < 7,
  Mu $> Lb,
  V1 $>= V0 * (1 + Lb),
  V1 $=< V0 * (1 + Mu),
  V1 $= ((1 + Mu) * V0) - DV,
  P $= (-1 / (4 * Mu)) * DV * (R1 - Rp),
  Zu $= (-1 / Mu),
  roundup(Zu,UpInt),  % round Zu up to the nearest integer
  Div $= UpInt - 1,
  Rest $= Zu - Div,
  P1 $= P / Zu,
  make_security_list(P1,Div,Rest,S),
  write('          Selling securities in period '),
  writeln(J).

make_security_list(P,D,Rest,[TheRest]) :-
  D $= 0,
  TheRest $= Rest * P.
make_security_list(P,Div,Rest,[P|TheRest]) :-
  Div $>= 1,
  Div0 $= Div - 1,
  make_security_list(P,Div0,Rest,TheRest).

add_two_security_lists([],Rest,Rest).
add_two_security_lists([H|T],[],[H|T]).
add_two_security_lists([First1|Rest1],
                       [First2|Rest2],[NewFirst|NewRest]) :-
  NewFirst $= First1 + First2,
  add_two_security_lists(Rest1,Rest2,NewRest).


/*****************************************************************
*             Small stuff                                        *
*****************************************************************/

/* Returns the maximum of two rational numbers */

cmax(X,Y,X) :- X $>= Y.
cmax(X,Y,Y) :- X $< Y.

/* Returns the minimum of two rational numbers */

cmin(X,Y,X) :- X $=< Y.
cmin(X,Y,Y) :- X $> Y.

/* X is a member of list */
```

```
member(X,[X|L]).
member(X,[Y|L]) :-
  member(X,L).

/* Test if there is security profit */

test_security_profit([],0,[]).
test_security_profit([First|Rest],First,Rest).

/* Compute the sum of list of rational numbers */

sum_list(List,Sum) :-
    sum_list(List,0,Sum).

sum_list([Head|Tail],Sum1,Sum3) :-
    Sum2 $= Head + Sum1,
    sum_list(Tail,Sum2,Sum3).
sum_list([],Sum,Sum).

/* Retrieve the NIIs from the list of results */

get_nii([],[]).
get_nii([results(_,Nii,_,_)|Rest],[Nii|NiiRest]) :-
    get_nii(Rest,NiiRest).


/*************************************************************
*  Maximization of Stability and Margin                      *
*************************************************************/

/* Warning: These predicates must be applied with care.
The order in which they appear in the clause body is important.
/*

/* Maximize stability
   L = [Nii0,Nii1,...,Niin]
   Up = upper bound for stability
*/

create_stability_constraint(L,Up) :-
    stability(S),
    create_S_constraint(L,S,Up,(1/1000)).

create_S_constraint(L,Low,Up,Precision) :-
    setval(sdata,[Low,Up]),
    repeat
       getval(sdata,[L0,U0]),
       Stability $= (U0 + L0) / 2,
       new_interval(L,L0,L1,U0,L1,Stability),
       setval(sdata,[L1,U1]),
       U0 - L0 $=< Precision,
    !.
```

```
new_interval(L,Low,Stab,Up,Up,Stab) :-
    test_bound(L,Stab),
    !.
new_interval(L,Low,Low,Up,Stab,Stab).

test_bound([_],_).
test_bound([Nii0,Nii1|Tail],Stability) :-
    Nii1 $>= Stability * Nii0,
    test_bound([Nii1|Tail],Stability).


/* Maximize the margin
   Marginlist is a list of the following structure

   [data(Nii1,TA1),data(Nii2,TA2),...,data(Niin,TAn)]

   TA is Total Assets
   Up is an upperbound of the margin.
   (p.s. this part is not used in the program)
*/

create_margin_constraint(MarginList,Up) :-
     margin(Margin),
     create_m_constraint(MarginList,Margin,Up,(1/100000)).

test_margin_bound([],_).
test_margin_bound([data(Nii,TA)|Tail],Margin) :-
     Nii $>= Margin * TA,
     test_margin_bound(Tail,Margin).

new_margin_bound(MarginList,Low,Margin,Up,Up,Margin) :-
     test_margin_bound(MarginList,Margin),
     !.
new_margin_bound(MarginList,Low,Low,Up,Margin,Margin).


create_m_constraint(MarginList,Low,Up,Precision) :-
   setval(mdata,[Low,Up]),
   repeat
     getval(mdata,[L0,U0]),
     Margin $= (U0 + L0) / 2,
     new_margin_bound(MarginList,L0,L1,U0,U1,Margin),
     setval(mdata,[L1,U1]),
     U0 - L0 $=< Precision,
   !.

/*************************************************************
*        Stuff which is used in the graphical part           *
*************************************************************/

map_rate(Scenario,Type,Nb,List) :-
```

```
    m_rate(Scenario,Type,1,Nb,List).

m_rate(Scenario,Type,N,N,[Rest]) :-
    rate(Scenario,Type,N,Rest).
m_rate(Scenario,Type,N0,N2,[Head|Tail]) :-
    N0 < N2,
    N1 is N0 + 1,
    rate(Scenario,Type,N0,Head),
    m_rate(Scenario,Type,N1,N2,Tail).

get_nth(1,[Head|Tail],Head).
get_nth(N1,[Head|Tail],Element) :-
        N1  > 1,
        N2 is N1 - 1,
        get_nth(N2,Tail,Element).

get_behavior(N,Results,[V,R]) :-
    cget_behavior(N,Results,V,R).

cget_behavior(_,[],[],[]).
cget_behavior(N,[results(_,_,B,C)|Tail],[V|Vrest],[R|Rrest]) :-
    get_nth(N,B,V),
    get_nth(N,C,R),
    cget_behavior(N,Tail,Vrest,Rrest).

length(List,N) :-
    length(List,0,N).

length([],N,N).
length([Head|Tail],N1,N3) :-
    N2 is N1 + 1,
    length(Tail,N2,N3).

maxval([],M,M).
maxval([Head|Tail],Max1,Max3) :-
     mmax(Max1,Head,Max2),
     maxval(Tail,Max2,Max3).

minval([],M,M).
minval([Head|Tail],M1,M3) :-
     mmin(M1,Head,M2),
     minval(Tail,M2,M3).

mmin(X,Y,X) :-
     X =< Y.
mmin(X,Y,Y) :-
     X > Y.
mmax(X,Y,X) :-
     X >= Y.
mmax(X,Y,Y) :-
     X < Y.
```

```
/*********************************************************************
************************ Graphical Part ****************************
*********************************************************************/


/* Make a nice graph of the interest rate scenario
The virtual coordinate system is set on
(LX,LY,UX,UY)=(0,0,2000,2000)
*/

map_scenario(X,Y,Size,Scenario,Nb) :-
    Size >= 600,
    map_rate(Scenario,ak2,Nb,Ak2),
/*  map_rate(Scenario,aibor,Nb,Aibor), */
    map_rate(Scenario,pdplus,Nb,Pdplus),
    map_rate(Scenario,pd,Nb,Pd),
    map_rate(Scenario,k,Nb,K),
    interior_style(0,1),
    line_type(0),
    X2 is X + Size - 100,
    Y2 is Y + Size - 100,
    X1 is X + 100,
    Y1 is Y + 100,
    line(X1,Y1,X1,Y2),
    line(X1,Y1,X2,Y1),
    YStep is fix((Y2 - Y1) / 15) - 1, % fix: truncate real to integer
    y_scale(X1,Y1,YStep,0,15),
    XStep is fix((X2 - X1) / Nb) - 1,
    x_scale(X1,Y1,XStep,0,Nb),
    mtext(X,Y2 + 50,'Percent'),
    mtext(X2 - 50,Y + 20,'Timesteps'),
    mtext(X1 + 40,Y2 + 50,'      INTEREST RATE SCENARIO'),
/*  make_rate_chart(X1,Y1,X2,Y2,XStep,YStep,0,aibor,Aibor), */
    make_rate_chart(X1,Y1,X2,Y2,XStep,YStep,1,pdplus,Pdplus),
    make_rate_chart(X1,Y1,X2,Y2,XStep,YStep,2,pd,Pd),
    make_rate_chart(X1,Y1,X2,Y2,XStep,YStep,3,k,K),
    make_rate_chart(X1,Y1,X2,Y2,XStep,YStep,4,ak2,Ak2).

y_scale(_,_,_,N,N).
y_scale(X,Y,Step,N1,N3) :-
    N1 < N3,
    N2 is N1 + 1,
    NewBase is Y + Step,
    mline(X - 3,Y + Step,X + 3,Y + Step),
    mtext(X - 90,Y + Step,N2),
    y_scale(X,NewBase,Step,N2,N3).

x_scale(_,_,_,N,N).
x_scale(X,Y,Step,N1,N3) :-
    N1 < N3,
    Base is X + Step,
    N2 is N1 + 1,
```

```
    stc(Y,Base,N2),
    x_scale(Base,Y,Step,N2,N3).

stc(Y,Base,N) :-
    member(N,[1,5,9,13,17,21,25,29,33]),
    !,
    mtext(Base,Y - 40,N),
    mline(Base,Y - 9,Base,Y + 3).
stc(Y,Base,N1) :-
    mline(Base,Y - 3,Base,Y + 3).

make_rate_chart(X1,Y1,X2,Y2,XS,YS,Type,Name,List) :-
    line_type(Type),
    Dots is Type * 40 + Y2 - 180,
    mline(X2 - 80,Dots,X2,Dots),
    mtext(X2 + 10,Dots,Name),
    compute_dots(List,X1,Y1,XS,YS,NewList),
    polyline(NewList).

compute_dots([],_,_,_,_,[]).
compute_dots([Head|Tail],X1,Y1,XS,YS,[Base,Y|Rest]) :-
    Base is X1 + XS,
    Y is fix(Head * YS * 100 + Y1) - 1,
    compute_dots(Tail,Base,Y1,XS,YS,Rest).

chart(X,Y,Size,minmaxbar,L) :-
     Size >= 500,
     line_type(0),
     X2 is X + Size - 100,
     Y2 is Y + Size - 100,
     X1 is X + 100,
     Y1 is Y + 100,
     mtext(X1,Y2 + 50,'          NET INTEREST INCOME'),
     line(X1,Y1,X1,Y2),
     line(X1,Y1,X2,Y1),
     maxval(L,0,Max),
     minval(L,10000,Min),
     Max > Min,
     Delta is (Size - 200) / (Max - Min),
     length(L,Nb),
     Nb >= 2,
     Step is fix((X2 - X1) / Nb),
     Smin is fix(1000 * Min) / 1000,
     mtext(X,Y1 - 20,Smin),
     Smax is fix(1000 * Max) / 1000,
     mtext(X,Y2 + 20,Smax),
     do_rest(L,Delta,Step,Min,X1,Y1).
chart(X,Y,Size,minmaxbar,LL).

do_rest([],_,_,_,_,_).
do_rest([Head|Tail],Delta,Step,Min,Base,Y1) :-
       Newbase is Base + Step,
```

```
        mrectangle(Base + 5,Y1,Newbase - 5,fix(Y1 + (Head - Min) * Delta)),
        do_rest(Tail,Delta,Step,Min,Newbase,Y1).

map_behavior(X,Y,Size,N,Results) :-
    Size >= 500,
    X1 is X + 100,
    X2 is X + Size - 100,
    Y1 is Y + 100,
    Y2 is Y + Size - 100,
    rectangle(X1,Y1,X2,Y2),
    integer_atom(N,NN),
    concat_atoms('   BS-ITEM ',NN,Title),
    mtext(X1,Y2 + 30,Title),
    get_behavior(N,Results,[V,R]),
    maxval(V,0,Vmax),
    minval(V,10000,Vmin),
    Vmax > Vmin,
    maxval(R,0,Rmax),
    minval(R,10000,Rmin),
    Rmax > Rmin,
    length(V,Nb),
    Nb >= 2,
    Rmin1 is fix(Rmin * 10000) / 100,
    Rmax1 is fix(Rmax * 10000) / 100,
    Vmin1 is fix(Vmin * 100) / 100,
    Vmax1 is fix(Vmax * 100) / 100,
    mtext(X2 + 10,Y2,Rmax1),
    mtext(X2 + 10,Y1,Rmin1),
    mtext(X + 10,Y2,Vmax1),
    mtext(X + 10,Y1,Vmin1),
    Xstep is fix((Size - 200) / (Nb - 1)),
    Vstep is (Size - 200) / (Vmax - Vmin),
    Rstep is (Size - 200) / (Rmax - Rmin),
    NNb is Nb - 1,
    x_scale(X1,Y1,Xstep,0,NNb),
    graph_minmax(X1,Y1,Xstep,Rstep,Rmin,R,Rdotlist),
    line_type(1),
    polyline(Rdotlist),
    graph_minmax(X1,Y1,Xstep,Vstep,Vmin,V,Vdotlist),
    line_type(0),
    polyline(Vdotlist).
map_behavior(X,Y,Size,N,Results).

graph_minmax(_,_,_,_,_,[],[]).
graph_minmax(X1,Y1,Xstep,Rstep,Rmin,[Head|Tail],[X1,Y|Rest]) :-
   NewX is X1 + Xstep,
   Y is fix(((Head - Rmin) * Rstep) + Y1),
   graph_minmax(NewX,Y1,Xstep,Rstep,Rmin,Tail,Rest).

mline(X1,Y1,X2,Y2) :-
      A1 is X1,
      B1 is Y1,
```

```
        A2 is X2,
        B2 is Y2,
        line(A1,B1,A2,B2).

mtext(X,Y,Text) :-
        A is X,
        B is Y,
        text(A,B,Text).

mrectangle(X1,Y1,X2,Y2) :-
        A1 is X1,
        B1 is Y1,
        A2 is X2,
        B2 is Y2,
        rectangle(A1,B1,A2,B2).


/*********************************************************
* Interest rate scenarios                                *
*                                                        *
* rate(Scenario_number, Interest_indicator, Time, Rate) *
*                                                        *
*********************************************************/

rate(S,ak2,J,R) :-
    rate(S,aibor,J,A),
    rate(S,k,J,K),
    R $= 0.5 * A + 0.5 * K.

rate(S,pd,J,R) :-
    rate(S,aibor,J,R).
rate(S,pdplus,J,R1) :-
    rate(S,aibor,J,R),
    R1 $= R + 0.005.


rate(1,aibor,J,0.06) :- J < 8.
rate(1,aibor,J,0.07) :- J > 7.

rate(2,aibor,J,0.06) :- J < 8.
rate(2,aibor,J,0.08) :- J > 7.

rate(3,aibor,J,0.07) :- J < 8.
rate(3,aibor,8,0.09).
rate(3,aibor,J,0.08) :- J > 8.

rate(4,aibor,J,0.068) :- J < 5.
rate(4,aibor,J,0.084) :- J > 4.

rate(5,aibor,J,0.068) :- J < 2.
rate(5,aibor,2,0.072).
rate(5,aibor,3,0.076).
```

```
rate(5,aibor,4,0.082).
rate(5,aibor,5,0.086).
rate(5,aibor,6,0.090).
rate(5,aibor,7,0.094).
rate(5,aibor,8,0.095).
rate(5,aibor,J,0.096) :- J > 8.

rate(6,aibor,0,0.06).
rate(6,aibor,1,0.06).
rate(6,aibor,2,0.06).
rate(6,aibor,3,0.06).
rate(6,aibor,4,0.065).
rate(6,aibor,5,0.075).
rate(6,aibor,6,0.080).
rate(6,aibor,7,0.085).
rate(6,aibor,8,0.080).
rate(6,aibor,9,0.075).
rate(6,aibor,10,0.065).
rate(6,aibor,11,0.06).
rate(6,aibor,J,0.06) :- J > 11.

rate(1,k,J,0.068) :- J < 5.
rate(1,k,J,0.070) :- J > 4.

rate(2,k,J,0.068) :- J < 5.
rate(2,k,J,0.080) :- J > 4.

rate(3,k,J,0.068) :- J < 5.
rate(3,k,J,0.11) :-  J > 4.

rate(4,k,J,0.068) :- J < 5.
rate(4,k,J,0.080) :- J > 4.

rate(5,k,J,0.068) :- J < 5.
rate(5,k,5,0.083).
rate(5,k,6,0.090).
rate(5,k,7,0.090).
rate(5,k,8,0.090).
rate(5,k,J,0.097) :- J > 8.

rate(6,k,J,0.068) :- J < 5.
rate(6,k,5,0.070).
rate(6,k,6,0.071).
rate(6,k,7,0.072).
rate(6,k,8,0.071).
rate(6,k,9,0.070).
rate(6,k,J,0.068) :- J > 9.

/************************ The End ************************/
```

# Bibliography


[1] K. Apt and M. Van Emden. **Contributions to the Theory of Logic Programming** Journal of the ACM, Vol. 29, No. 3, 1982.

[2] F. Berthier. **Using CHIP to Support Decision Making.** in *Actes du Seminaire 1988 - Programmation en Logique,* Tregastel (France), 1988.

[3] F. Berthier. **Managing Underlying Assumptions of a Financial Planning Model in CHIP.** Technical Report TR-LP-39, ECRC, 1988.

[4] R. W. Brown, W. D. Northup and J. F. Shapiro. **LOGS: A modeling and optimization system for business planning.** in *Computer Assisted Decision Making.* C. Mitra (ed.). Elseviers Science Publishers, 1986.

[5] A. Colmerauer. **Note Sur Prolog III.** in *Actes du Seminaire 1986 - Programmation en Logique.* Tregastel (France), 1986.

[6] M. Dincbas and J-P. Lepape. **Metacontrol of Logic Programs in METALOG.** in *Proceedings of the International Conference On Fifth Generation Computer Systems* (FGCS-84). Tokyo, 1984.

[7] M. Dincbas and P. Van Hentenryck. **Constraints and Logic Programming.** Technical Report TR-LP-9, ECRC, 1986.

[8] M. Dincbas, P. van Hentenryck, H. Simonis, A. Aggoun, T. Graf and F. Berthier. **The Constraint Logic Programming Language CHIP.**, in *Proceedings of the International Conference on Fifth Generation Computer Systems*, Tokyo, 1988.

[9] M. Dincbas, H. Simonis and P. Van Hentenryck. **Solving the Car Sequencing Problem in Constraint Logic Programming.** in *European Conference on Artificial Intelligence* (ECAI-88)., Munich (W. Germany), 1988.

[10] D. Gilbert. **The Tools of Interest Rate Risk Management.** in *Managing Bank Assets and Liabilities.* J. Wilson (ed.). Euromoney Publications, 1988.

[11] T. Graf, P. Van Hentenryck, C. Pradelles and L. Zimmer. **Simulation of Hybrid Circuits in Constraint Logic Programming.** in *International Joint Conference on Artificial Intelligence.* Detroit, 1989.

[12] P. Van Hentenryck and M. Dincbas. **Forward Checking in Logic Programming.** in *Fourth International Conference on Logic Programming*, Melbourne (Australia), 1987.

[13] P. Van Hentenryck and J-P. Carillion. **Generality versus Specifity: an Experience with AI and OR techniques.** AAAI-88, 1988.

[14] P. Van Hentenryck. **Constraint Satisfaction in Logic Programming.** Logic Programming Series. The MIT Press, Cambridge MA, 1989.

[15] M.C. Huizer. **The ALM Function: Development and Strategy.** in *Managing Bank Assets and Liabilities.* J. Wilson (ed.). Euromoney Publications, 1988.

[16] J. Jaffar and J-L. Lassez. **Constraint Logic Programming.** in *Proceedings of the 14th ACM POPL Symposium.* Munich (W. Germany), 1987.

[17] J. Jaffar and S. Michaylov. **Methodology and Implementation of A CLP system.** in *Fourth International Conference on Logic Programming.* Melbourne (Australia), 1987. (MIT Press.)

[18] J.P.C. Kleijnen. **Statistical design and analysis techniques.** in *Progress in Modelling and Simulation.* F.E. Cellier (ed.). Academic Press, 1982.

[19] M. I. Kusy and W. T. Ziemba. **A Bank Asset and Liability Management Model.** Operations Research, Vol. 34, No. 3, 1986.

[20] G. D. Latainer. **The Term Structure of Interest Rates.** in *Controlling Interest Rate Risk.* R. Platt (ed.). John Wiley & Sons, 1986.

[21] J. W. Lloyd. **Foundations of Logic Programming.** Second Edition, Springer-Verlag, 1987.

[22] L. Naish. **Automating Control for logic programs.** Journal of Logic Programming, 3:167-183, 1985.

[23] L. Naish. **Negation and Control in Prolog.** PhD Thesis. University of Melbourne (Australia), 1985.

[24] F. W. M. Pallada. **Rentemarge en renterisico's bij financiele instellingen.** Rotterdamse monetaire studies, nr. 25, Rotterdam, 1987.

[25] R. B. Platt. **Controlling Interest Rate Risk.** in *Controlling Interest Rate Risk.* R. B. Platt (ed.). John Wiley & Sons, 1986.

[26] A. Schrijver. **Theory of Linear and Integer Programming.** Wiley-Interscience Series in Discrete Mathematics, 1986.

[27] E. Shapiro. Review of J. Lloyd's **Foundations of logic programming.** Computing Reviews, 8608-0668, 1986.

[28] H. A. Simon. **The Sciences of The Artificial.** Second Edition. MIT Press, Cambridge MA, 1981.

[29] H. Simonis, N. Nguyen and M. Dincbas. **Verification of Digital Circuits Using CHIP.** IFIP, 1988. (Elseviers Science Publishers)

[30] A. Toevs and W. Haney. **Measuring and Managing Interest Rate Risk: A guide to Asset/Liability Models Used in Banks and Thrifts.** in *Controlling Interest Rate Risk.* R. Platt (ed.). John Wiley & Sons, 1986.

ITK: P.O. BOX 90153 5000 LE TILBURG THE NETHERLANDS